Simulation and Design of Quantum Circuits

Alwin Zulehner and Robert Wille(✉)

Institute for Integrated Circuits, Johannes Kepler University Linz, Linz, Austria
{alwin.zulehner,robert.wille}@jku.at

Abstract. Currently, there is an ongoing "race" to build the first practically useful quantum computer that provides substantial speed-ups for certain problems compared to conventional computers. In addition to the development of such devices, this also requires the development of automated tools and methods that provide assistance in the simulation and design of corresponding applications. Otherwise, a situation might be reached where we have powerful quantum computers but hardly any proper means to actually use them. This work provides an overview of corresponding solutions for the task of quantum circuit simulation, the task of quantum circuit design, as well as corresponding mapping tasks. The covered solutions utilise expertise on efficient data structures and algorithms gained in the design of conventional circuits and systems over the last decades. While the respective descriptions are kept brief and mainly convey the general ideas, references to further readings are provided for a more detailed treatment.

1 Introduction

In quantum computing, so-called quantum bits (i.e., qubits) serve as elementary information unit, which—in contrast to conventional bits—can not only be in one of its two orthogonal basis states (denoted $|0\rangle$ and $|1\rangle$ using Dirac notation), but also in superposition (i.e., a linear combination) of both [1]. Together with further quantum-physical phenomena such as entanglement (the state of a qubit might be influenced by the state of other qubits), this allows that the pure state of a quantum system composed of n qubits may represent a superposition of 2^n basis states and corresponding complex amplitudes—resulting in higher information density and computational power.

Well-known initial representatives of quantum algorithms following this powerful computation paradigm are Grover's search algorithm [2] and Shor's algorithm for integer factorisation in polynomial time [3]—both allowing to significantly outperform conventional machines. Recently, the application area of quantum algorithms has significantly broadened and provides efficient methods in areas like chemistry, solving systems of linear equations, physics simulations, machine learning, and many more [4–6].

These developments are also triggered by the fact that quantum computers are reaching feasibility since "big players" such as *IBM*, *Google*, *Microsoft*, and

I. Ulidowski et al. (Eds.): RC 2020, LNCS 12070, pp. 60–82, 2020.
https://doi.org/10.1007/978-3-030-47361-7_3

Intel as well as specialised startups such as *Rigetti* and *IonQ* have entered this research field and are heavily investing in it [7–11]. In 2017, this led to the first quantum computers that are publicly available through cloud access by IBM. Since then, their machines have been used by more than 100,000 users, who have run more than 6.5 million experiments thus far. Recently, IBM followed with the presentation of their prototype towards a quantum computer for commercial use (a stand-alone quantum computer to be operated outside of their labs)—the *IBM Q System One* presented in January 2019 at CES [12].

Since currently available quantum computers are still limited in the number of qubits, gate fidelity, as well as coherence time, they are classified as *Noisy Intermediate Scale Quantum* (NISQ [5]) devices that will only be able to successfully run some of the quantum algorithms outlined above (due to their limitations). In fact, unveiling the full potential of quantum computing requires—besides further reduction of error rates and improvement of coherence time—error-correcting codes where each logical qubit in a computation is realised by several (up to several hundreds) of physical qubits—eventually resulting in *fault-tolerant* devices that are capable of conducting very deep computations on a large number of qubits and with perfect accuracy [13,14].

In addition to these accomplishments and prospects, also the development of automated tools and methods that provide assistance in the simulation and design of corresponding applications is required. In this regard, the task of quantum circuit simulation, the task of quantum circuit design, as well as corresponding mapping tasks are important. Since modelling (arbitrary) quantum states on conventional machines requires exponential overhead and many design problems are of exponential nature, straightforward solutions for these tasks will not scale to relevant problem sizes. Hence, clever data-structures and algorithms are required that allow for efficient solutions (at least) in certain cases. Otherwise, we are approaching a situation where we might have powerful quantum computers but hardly any proper means to actually use them.

This work provides an overview on solutions which have been developed for these tasks and utilise expertise on efficient data structures and algorithms gained in the design automation community over the last decades for conventional circuits and systems. To this end, the simulation of quantum circuits, their design, as well as technology mapping (compiling) are covered and discussed from a design automation perspective. The reviewed solutions often yield improvements of several orders of magnitude compared to the current state of the art (regarding runtime and corresponding design objectives)—showing the tremendous available potential.

The overview is thereby structured as follows: First, Sect. 2 provides a background on quantum computing. Afterwards, Sect. 3, Sect. 4, and Sect. 5 sketch the developed methods for the considered design tasks, i.e., quantum-circuit simulation, the design of Boolean components occurring in quantum algorithms, as well as mapping quantum circuits to real hardware (including references to further reading for a more detailed treatment). Finally, Sect. 6 concludes the paper.

2 Background on Quantum Computing

Quantum computations operate on *qubits*—two-level quantum systems that can be combined into n-qubit systems. The state of a qubit is given by a linear combination (i.e., a *superposition*) of these basis states $|\varphi\rangle = \alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$, where the complex *amplitudes* α_0 and α_1 satisfy $\alpha_0\alpha_0^* + \alpha_1\alpha_1^* = 1$.

The joint state of n qubits (also denoted as the system's *wave function*) is contained in the *tensor product* of n two-dimensional Hilbert spaces—the 2^n-dimensional Hilbert space spanned by the basis $|0\rangle, \ldots, |2^n - 1\rangle$. Hence, a *superposition* of all computational basis states may need up to 2^n complex-valued parameters—appearing as the amplitudes of the unit-norm state vector.

Definition 1. *Consider a quantum system composed of n qubits. Then, all possible states of the system are of the form*

$$|\varphi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x \cdot |x\rangle\,, \text{ where } \sum_{x \in \{0,1\}^n} \alpha_x \alpha_x^* = 1 \text{ and } \alpha_x \in \mathbb{C}.$$

The state $|\varphi\rangle$ *can be also represented by a column vector* $\varphi = [\varphi_i]$ *with* $0 \leq i < 2^n$ *and* $\varphi_i = \alpha_x$*, where* $nat(x) = i$*.*

Quantum states cannot be directly observed. To extract (partial) information from quantum states in the form of conventional bits, one performs a *measurement operation*. In contrast to conventional computers, this measurement modifies the quantum state. In the process of measurement, the quantum state non-deterministically collapses to one of these basis states where the probability of each outcome reflects the proximity to the respective basis state. More precisely, measuring a one-qubit state $\alpha_0 \cdot |0\rangle + \alpha_1 \cdot |1\rangle$ (with $\alpha_0\alpha_0^* + \alpha_1\alpha_1^* = 1$) changes the state to $|0\rangle$ or $|1\rangle$ with probabilities $\alpha_0\alpha_0^*$ and $\alpha_1\alpha_1^*$, respectively.

Example 1. *Consider a quantum system composed of* $n = 3$ *qubits* q_0*,* q_1*, and* q_2 *that assumes the state* $|\varphi\rangle = |q_0 q_1 q_2\rangle = \frac{1}{2} \cdot |010\rangle + \frac{1}{2} \cdot |100\rangle - \frac{1}{\sqrt{2}} \cdot |110\rangle$*. Then, the state vector of the system is given by*

$$\varphi = \left[0, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, -\frac{1}{\sqrt{2}}, 0\right]^T.$$

Measuring the system yields basis states $|010\rangle$*,* $|100\rangle$*, and* $|110\rangle$ *with probabilities* $\frac{1}{4}$*,* $\frac{1}{4}$*, and* $\frac{1}{2}$*, respectively. Measuring only qubit* q_0 *collapses* q_0 *into basis state* $|0\rangle$ *and* $|1\rangle$ *with probabilities* $\frac{1}{4}$ *and* $\frac{1}{4} + \frac{1}{2} = \frac{3}{4}$*, respectively—changing the state of the system either to* $|\varphi'\rangle = |010\rangle$ *or to* $|\varphi''\rangle = \frac{1}{\sqrt{3}} \cdot |100\rangle - \sqrt{\frac{2}{3}} \cdot |110\rangle$*.*

Aside from measurements, quantum computers apply quantum operations to a fixed set of qubits, altering the joint state of the qubits in a reversible fashion. These operations are described by unitary matrices of size $2^n \times 2^n$. Simple quantum operations (also denoted *gates*) are defined over one or two qubits only. Mathematically speaking, the resulting $2^n \times 2^n$ matrix can then

be computed as the Kronecker product of the matrix representing the gate's operation and a large identity matrix.

Commonly used quantum gates for generating a superposition (the Hadamard operation H), inverting a quantum state (X), and applying phase shifts by -1 (Z), are respectively defined as

$$\mathrm{H} = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \mathrm{NOT} = \mathrm{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \mathrm{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Two-qubit gates can couple pairs of qubits and are represented by 4×4 unitary matrices. By applying arbitrary two-qubit gates to different pairs of qubits, it is possible to effect any 2^n-dimensional unitary, i.e., attain universal quantum computation (each quantum functionality can be realised with those gates). It is common to allow a variety of one-qubit gates but limit two-qubit gates, e.g., to CNOT gates:

$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The two-qubit CNOT gate can also be defined by its action $|x\ y\rangle \mapsto |x\ x \oplus y\rangle$, where $\oplus$ represents the *exclusive-or* (XOR) operation, the unmodified qubit x is called *control*, and the other bit is called *target*.

Quantum circuits [1] are used as proper description means for a finite sequence of "small" gates that cumulatively enact some unitary operator U and, given an initial state $|\varphi\rangle$ (which is usually the basis state $|0 \ldots 0\rangle$), produce a final state vector $|\varphi'\rangle = |U\varphi\rangle$. Hence, a quantum gate does not represent a physical entity (like in the conventional realm), rather an operation that is applied to a set of qubits.

Definition 2. *In quantum circuits, the qubits are vertically aligned in a circuit diagram, and the time axis (read from left to right) is represented by a horizontal line for each qubit. Boxes on the time axis of a qubit (or enclosing several qubits) indicate gates to be applied.*[1] *Note that measurement also counts as quantum operation in this context. Control qubits are indicated by* $\bullet$ *and are connected to the controlled operations by a single line.*

Example 2. *Figure 1 shows a quantum circuit. The circuit contains two qubits,* q_0 *and* q_1*, which are both initialised with basis state* $|0\rangle$*. First, a Hadamard operation is applied to qubit* q_0*, which is represented by a box labelled* H*. Then, a* CNOT *operation is conducted, where* q_0 *is the* control qubit *(denoted by* $\bullet$*) and* q_1 *is the target qubit (denoted by* $\oplus$*). Eventually, qubit* q_0 *is measured as indicated by the meter symbol.*

When two gates are applied on the same qubits in sequence, the resulting operation is represented by the matrix product of gate matrices. When an

[1] Note that an X gate may also be denoted by $\oplus$.

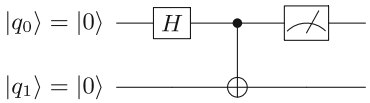


Fig. 1. Quantum circuit.

m-qubit gate A and an n-qubit gate B are applied in parallel (on different qubits), the resulting operation is represented by the *Kronecker product* $A \otimes B$ of two matrices.

Example 3. *Consider again the quantum circuit shown in Fig. 1. The resulting state* $|\varphi'\rangle$ *(before measurement) is determined by multiplying the respective unitary matrices to the state vector. Since the Hadamard gate shall only affect* q_0*, the Kronecker product of* H *and the identity matrix* I_2 *is formed, i.e.,*

$$H \otimes I_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}.$$

Then, $|\varphi'\rangle$ *is determined by*

$$|\varphi'\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

As can be seen, the two gates entangle the qubits q_0 *and* q_1 *—generating a so-called* Bell state $|\varphi'\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$*. Measuring qubit* q_0 *collapses its superposition into one of the two basis states. Since* q_0 *and* q_1 *are entangled,* q_1 *collapses to the same basis state.*

3 Quantum-Circuit Simulation

Since physical realisations of quantum computers are limited in their availability, their number of qubits, their gate fidelity, and coherence time, quantum-circuit simulators running on conventional machines are required for many tasks. From a user's perspective, possible applications (or at least their prototypes) for quantum computers are usually first evaluated through simulators that serve as temporary substitute. Moreover, simulation can be adapted to circuit equivalence-checking and other functional verification tasks useful for circuit designers [15–17]. Simulation also plays an important role for designers of quantum systems, e.g., to foster the development of error-correcting codes. Besides that, the urgent need of verifying quantum hardware might be conducted (at least some of the required verification tasks) by comparing runs on these machines to simulation

outcome [18,19]. Ultimately, quantum-circuit simulation capabilities provide an estimate on *quantum supremacy* [18] as well as to identify classes of circuits where no quantum speed-up is reachable (i.e., in case these circuits can be simulated efficiently on a conventional machine). In all these scenarios, simulators may give additional insights since, e.g., the precise amplitudes of a quantum state are explicitly determined (while they are not observable in a real quantum computer).

However, quantum-circuit simulation in general constitutes a computationally very complex task since each quantum gate and each quantum state is eventually represented by a unitary matrix or state vector that grows exponentially with the number of qubits. In fact, each quantum operation applied to a quantum state composed of n qubits requires multiplying a $2^n \times 2^n$-dimensional matrix with a 2^n-dimensional vector.[2] This constitutes a serious bottleneck, which prevents the simulation of many quantum applications and, by this, the evaluation of their potential. In fact, the array-like representation of the state vector in current state-of-the-art simulators limits the number of qubits to be simulated to approximately 30 on a modern computer (and to 50 when considering supercomputers with petabytes of distributed memory) [20].

This section presents a complementary simulation approach that aims for overcoming this memory bottleneck (based on [21]). To this end, dedicated *Decision Diagrams* (DDs) are developed, which reduce the memory requirements by representing redundancies in the occurring vectors and matrices by means of shared nodes. This allows gaining significant improvements compared to straightforward realisations (relying on array-like representations) in many cases—often reducing the simulation time from several hours or days to seconds or minutes.[3]

3.1 General Idea

The general idea of the presented complementary approach is to exploit redundancies in the 2^n-dimensional vectors representing quantum states. To this end, decision diagram techniques (similar to those from the conventional realm) are employed. More precisely, a given state vector with entries being complex numbers is decomposed into sub-vectors. To this end, consider a quantum system with qubits $q_0, q_1, \ldots q_{n-1}$, whereby without loss of generality q_0 represents the most significant qubit. Then, the first 2^{n-1} entries of the corresponding state vector represent the amplitudes for the basis states with q_0 set to $|0\rangle$; the other entries represent the amplitudes for states with q_0 set to $|1\rangle$. This decomposition is represented in a decision diagram structure by a node labelled q_0 and two successors leading to nodes representing the sub-vectors. The sub-vectors are

[2] Note that different simulation approaches exist that do not compute the complete final state vector, and that it is usually not necessary to represent the exponentially large matrix explicitly. However, this does not decrease the exponential complexity.

[3] Note that previous DD-based simulators (e.g., *QuIDDPro* [22]) did not get established due to their limited applicability (i.e., they provide improvements in rather few cases).

recursively decomposed further until vectors of size 1 (i.e., a complex number) result. This eventually represents the amplitude α_i for the basis state and is given by a terminal node. During these decompositions, equivalent sub-vectors are represented by the same node—allowing for sharing and, hence, a reduction of the memory complexity. An example illustrates the idea.

Example 4. *Consider a quantum system with $n = 3$ qubits situated in a state given by the following vector:*

$$\varphi = \left[0, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, -\frac{1}{\sqrt{2}}, 0\right]^T.$$

Applying the decompositions described above yields a decision diagram as shown in Fig. 2a. The left (right) outgoing edge of each node labelled q_i points to a node representing the sub-vector with all amplitudes for the basis states with q_i set to $|0\rangle$ ($|1\rangle$). Following a path from the root to the terminal node yields the respective entry. For example, following the path highlighted bold in Fig. 2a provides the amplitude for the basis state with $q_0 = |1\rangle$ (right edge), $q_1 = |1\rangle$ (right edge), and $q_2 = |0\rangle$ (left edge), i.e., $-\frac{1}{\sqrt{2}}$ which is exactly the amplitude for basis state $|110\rangle$ (seventh entry in the vector). Since some sub-vectors are equal (e.g., $\left[\frac{1}{2}, 0\right]^T$ represented by the left node labelled q_2), sharing is possible.

However, even more sharing is possible since sub-vectors often differ in a common factor only. This is additionally exploited in the proposed representation by denoting common factors of amplitudes as weights attached to the edges of the decision diagram. Then, the value of an amplitude for a basis state is determined by following the path from the root to the terminal, and additionally multiplying the weights of the edges along this path. Again, an example illustrates the idea.

Example 4 (continued)**.** *As can be seen, the sub-vectors represented by the nodes labelled q_2 (i.e., $\left[\frac{1}{2}, 0\right]^T$ and $\left[-\frac{1}{\sqrt{2}}, 0\right]^T$) differ in a common factor only.*

In the decision diagram shown in Fig. 2b, both sub-trees are merged. This is possible since the corresponding value of the amplitudes is now determined not by the terminals, but the weights on the respective paths. As an example, consider again the path highlighted bold representing the amplitude for the basis state $|110\rangle$. Since this path includes the weights $\frac{1}{2}$, 1, $-\sqrt{2}$, and 1, an amplitude of $\frac{1}{2} \cdot 1 \cdot (-\sqrt{2}) \cdot 1 = -\frac{1}{\sqrt{2}}$ results.

Note that, of course, various possibilities exist to factorise an amplitude. Hence, a normalisation is applied which assumes the left edge to inherit a weight of 1. More precisely, the weights w_l and w_r of the left and right edge are both divided by w_l and this common factor is propagated upwards to the parents of the node. If $w_l = 0$, the node is normalised by propagating w_r upwards to the parents of the node.

The idea used for representing state vectors by means of DDs can be extended to also represent unitary matrices. Here, each DD-node has four successors that

(a) Without edge weights. (b) With edge weights.

Fig. 2. DD-based representation of state vectors.

represent the four quadrants of the sub-matrix. Having description means for state vectors and unitary matrices (describing the functionality of gates) it is left to provide algorithms for matrix-vector multiplication as well as for measurement. Fortunately, all these operations can be directly employed on the DDs and without the need of explicitly representing the underlying exponentially large entities. For further details we refer to [21].

3.2 Resulting Approaches

Following the general idea outlined above leads to a simulation approach that scales polynomially with the size of the DD representing the state vector. Since the DD often remains rather compact, significant improvements can be observed compared to straightforward Schrödinger-style simulators as well as to previous DD-based simulators in many cases—even though these techniques have been heavily optimised over the last decade and utilise multiple CPU-cores to reduce simulation time (while the proposed approach utilises a single core only). More precisely, the approach proposed in [21] is capable of (1) simulating quantum computations for more qubits than before, (2) in significantly less run-time, and (3) on a regular Desktop machine.

For further details on the basic ideas and required algorithms of the DD-based simulator we refer to [21,23]. Moreover, [21] shows that for many cases, the simulation time can be reduced from several days to just a few seconds or minutes. This initial version of a DD-based simulator did not only lead to a significant improvement compared to the current state of the art, but has also received significant acknowledgement by the community—triggering further optimisations as done for array-based Schrödinger-style simulators for more than a decade.

Using DDs for representing occurring vectors and matrices, the complexity of multiplications depends on the size (i.e, the number of nodes) of the respective operands in DD-based simulation. Together with the fact that the DDs for the usually considered gate matrices are linear in size (with respect to the number

of qubits), this implies that it might be beneficial to combine gate operations before applying them to the state vector. In [24], strategies are described for combining operations that allow improving the initial version of the proposed DD-based simulator significantly—up to several orders of magnitude when exploiting application-specific knowledge.

Enormous improvements compared to the state of the art as described above obviously require an efficient implementation of the underlying DD-package—especially for handling the occurring complex numbers. By providing such techniques—in joint consideration of implementation techniques for decision diagrams in the conventional domain developed decades ago—the development of a powerful DD-package for the quantum domain was leveraged in [25]. The evaluation conducted in [25] showed that complex numbers can be handled much more efficiently than in previous implementations and that decision diagrams for established quantum functionality is constructed in significantly less run-time (up to several orders of magnitude). Presumably, this performance boost can be easily passed to DD-based methods for other design automation tasks like synthesis [26,27] or verification [15–17], just by incorporating this new package.

Since handling complex numbers is crucial in DDs for quantum computation (especially when occurring as edge weights), the resulting trade-off between accuracy and compactness has been thoroughly discussed and evaluated in [28]. Since this trade-off requires fine-tuning of parameters on a case-by-case basis and might still yield useless results, an algebraic decision diagram is proposed in [28] to overcome this issue. The proposed algebraic representation guarantees perfect accuracy while remaining compact (all redundancies that are actually present are detected)—with moderate overhead in many cases.

All the endeavours listed above have been implemented in C/C++ and made publicly available at http://iic.jku.at/eda/research/quantum_simulation. Besides that, a stand-alone version of the developed DD-package is available at http://iic.jku.at/eda/research/quantum_dd. Together with the significant improvements gained compared to the state of the art, this did not only result in acknowledgement inside the academic community, but also received interest from big players in the field. More precisely, the developed simulation approach has been acknowledged with a *Google Research Faculty Award* and has recently been officially integrated into IBM's SDK *Qiskit*. This further emphasises the potential of DD-based design methods in the quantum domain—hopefully leading to as powerful DD-based methods as taken for granted in the conventional domain today. Questions on whether hybrid approaches are possible or whether concurrent approaches as well as approximation schemes can be exploited remain open issues for future work. First results towards these questions are provided in [29,30].

4 Design of Boolean Components for Quantum Circuits

Estimating resource requirements of quantum algorithms (i.e., the number of required qubits and run-time on quantum computers), their simulation, or their

execution on real hardware requires compiling quantum algorithms containing high-level operations (e.g., modular exponentiation in Shor's algorithm) into quantum circuits composed of elementary gates available on the considered target architecture. Thereby, quantum circuits composed of gates with multiple control qubits (multiple-controlled qubit gates) are usually considered since they (1) describe a rather low-level but still technology independent description of the algorithm, (2) can be directly handled by most simulators, and (3) are usually utilised as input for technology mapping algorithms (which will be covered in the next section).

For the "quantum part" of an algorithm, a decomposition into multiple-controlled qubit gates is usually inherently given by the algorithm, by using common building blocks like a *Quantum Fourier Transform* (QFT [31]), or determined by hand. However, this is different for large Boolean components that are contained in many quantum algorithms, e.g., the modular exponentiation in Shor's algorithm for integer factorisation [3] or a Boolean description of the database that is queried in Grover's algorithm [2].

Even though the functionality of the Boolean components can be described in the conventional domain, corresponding design methods cannot be utilised since the inherent reversibility of quantum computations has to be considered. In fact, determining circuits composed of reversible gates only, requires dedicated *reversible-circuit synthesis* approaches. To manage the complex functionality of Boolean components, they are usually split into several (non-)reversible parts [32]. However, these resulting non-reversible sub-functions have to be *embedded* into reversible ones to ensure the desired unique mapping from inputs to outputs—a task that can either be conducted explicitly or implicitly. This embedding process requires adding several so-called *ancillary qubits*, which shall be kept as small as possible since qubits are a highly limited resource. Besides that, T-count and T-depth of the synthesised reversible circuits serve as cost metric to compare different approaches that yield circuits with an equal (or at least a close-to equal) number of qubits.

This section focuses on the *functional* design flow for synthesising Boolean components (where the reversible function resulting from an explicit embedding step is passed to synthesis algorithm) since it yields circuits with a moderate number of qubits (often the minimum). Investigating this problem from a design automation perspective allows developing efficient methods utilising the decision diagrams introduced in the context of simulation (cf. Sect. 3) [33–35]. However, there is even more (yet) unused potential that allows synthesising cheaper circuits, yields better scalability, and even reduces the number of required qubits below what is currently considered as the minimum (for certain cases)—significantly improving the current state of the art.

4.1 One-Pass Design of Reversible Circuits

Despite using efficient description means like DDs for functional synthesis, the currently established design flow still suffers from the need to conduct embedding

and actual synthesis separately—a major drawback that prohibits the exploitation of a huge degree of freedom since embedding is not necessarily conducted in a fashion, which suits the following synthesis step. To overcome this drawback, the work [36,37] introduced a completely new design flow that combines functional synthesis and the embedding to a *one-pass design flow*. This generic flow is not bound to a certain functional synthesis approach and—for the first time—exploits the available degree of freedom to significantly increase scalability and to reduce the costs of the synthesised circuit while keeping the number of required qubits at the minimum.

In the established flow, an individual step is required that embeds the non-reversible function to be synthesised into a reversible one. Thereby, $k = \lceil \log_2 \mu(p_1) \rceil$ further so-called garbage outputs are added (assuming that the most frequent output pattern p_1 occurs $\mu(p_1)$ times) and the additional rows and columns of the truth table are assigned such that a unique mapping from inputs and outputs results [33]. Passing a non-reversible function directly to a functional reversible-circuit synthesis approach will fail, since several input combinations shall be mapped to the same output combination. This can be avoided in two ways:

- Following the *exact* solution guarantees to result in a circuit requiring the minimum number of qubits. The general idea is to add k further variables to the function description (e.g., a DD), but keep all additional entries in the function *don't care*—allowing to exploit the available degree of freedom of their assignment (which does not matter as long as a reversible function results). Having these additional variables allows conducting synthesis (almost) as usual. During synthesis, the *don't cares* are inherently assigned (1) in a way that suits best to the synthesis algorithm, and (2) such that a reversible function results (since only reversible gates are added to the circuit).
- Following the *heuristic* solution does not necessarily result in a circuit requiring the minimum number of qubits, but still bounded. The general idea is to conduct synthesis without embedding. Whenever an error is encountered during synthesis (i.e., synthesis cannot proceed due to the missing embedding step), the function to be synthesised is modified such that the algorithms can continue. Since this obviously results in a circuit different to the intended one, the modifications of the function are stored on so-called buffer-lines (at most one buffer line is required for each variable of the function). After synthesis finishes, these modifications are reverted by a single CNOT gate for each buffer line.

  The advantage of the heuristic approach is that no additional variables are added to the function description (as done in the usual functional design flow and the exact one-pass design). Hence, this heuristic approach is even more scalable that the exact solution since the function description remains smaller.

Example 5. *Consider a function* $f : \mathbb{B}^n \to \mathbb{B}^m$ *with* n *inputs and* m *outputs and assume that the most frequent output pattern occurs* $\mu(p_1)$ *times. Then,*

following the exact solution, the f is enriched by $k = \lceil \log_2 \mu(p_1) \rceil$ further outputs to make all output patterns distinguishable. Hence, the synthesis is conducted on a function with $\max(n, m + k)$ *variables—like in the established design flow. However, the additional entries in the truth table remain* don't care *initially and are assigned 0 or 1 during synthesis as suitable.*

Instead, the heuristic solution conducts synthesis directly on f and, hence $\max(n, m)$ *variables. The modifications made to f during synthesis require at most* $\min(n, m)$ *buffer lines—resulting in a quantum circuit with at most $n + m$ qubits.*

The evaluations provided in [36] show the advantages of the one-pass design flow (which can be also applied to other functional synthesis approaches) compared to the conventional two-stage design flow. Besides substantial speedups compared to the state-of-the-art design flow, the T-count is reduced by several orders of magnitude in most cases—clearly outperforming the currently established functional design flow for reversible circuits where embedding and synthesis are conducted separately. For further details, we refer to [36].

4.2 Exploiting Coding Techniques

The proposed one-pass design flow can be enriched with the idea of exploiting coding techniques in order to reduce the number of variables that have to be considered during synthesis [38].[4] This idea is based on the fact, that the output patterns in non-reversible functions are not uniformly distributed—leading to a situation where some patterns require many additional outputs while others require only a few. Hence, several garbage outputs are required only for certain output patters. Avoiding this overhead provides significant potential for improving synthesis. In fact, employing a variable-length code allows realising any non-reversible function with a single ancillary qubit only—allowing conducting synthesis on significantly fewer variables than before [39]. The key idea is to represent frequently occurring output patterns (which require more garbage outputs) with a smaller number of variables. Vice versa, less frequently occurring patterns (which require less garbage outputs) are represented with a larger number of variables. In other words, coding techniques are utilised in order to encode the desired function with a variable-length code in which the length of the code word for an output pattern p_i is indirectly proportional to the number $\mu(p_i)$ of times the pattern occurs. An example illustrates that.

Example 6. *Consider the Boolean function shown in Table 1a and its distribution of the output patterns as shown in Table 1b. Following, e.g., the exact one-pass design flow outlined above results in a function with 5 inputs/outputs since the most frequent output pattern $p_1 = 010$ occurs four times and, thus, requires two garbage outputs. However, using a variable-length code as shown in*

[4] Note that exploiting coding techniques is also possible in the original design flow composed of an embedding and a synthesis step.

Table 1. Variable-length encoding for one-pass design.

(a) Orig. function

x_0	x_1	x_2	y_0	y_1	y_2
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	1

(b) Output patterns

i	p_i	$\mu(p_i)$
1	010	4
2	100	2
3	001	1
4	011	1
5	000	0
6	101	0
7	110	0
8	111	0

(c) Encoding

i	p_i	$c(p_i)$
1	010	0 - -
2	100	1 0 -
3	001	1 1 0
4	011	1 1 1

(d) Encoded function

x_0	x_1	x_2	y_0	y_1	y_2
0	0	0	0	-	-
0	0	1	0	-	-
0	1	0	1	0	-
0	1	1	1	0	-
1	0	0	1	1	1
1	0	1	0	-	-
1	1	0	0	-	-
1	1	1	1	1	0

Table 1c allows reducing the number of required qubits. There, the most frequent output pattern is encoded by $c(p_1) = 0$. Since this pattern requires two garbage outputs, in total $1+2 = 3$ outputs are required.[5] *The second most frequent output pattern $p_2 = 100$ is encoded by $c(p_2) = 10$. Since this pattern occurs only twice, one garbage output is required—again resulting in $2 + 1 = 3$ outputs. The patterns p_3 and p_4 are encoded by $c(p_3) = 110$ and $c(p_4) = 111$, respectively. Here, no garbage outputs are required. The remaining patterns (p_5 to p_8) do not have to be encoded, since they never occur. Overall, this yields an (encoded) reversible function which embeds f as shown in Table 1d and is composed of a total of 3 inputs/outputs only—two qubits fewer than without using coding.*

The code is computed by generating a *Pseudo-Huffman* tree: Starting with terminal nodes—one for each output pattern with $\mu(p_i) > 0$ (no code has to be assigned to output patterns that do not occur)—with attached weights representing the number of respectively required garbage outputs (i.e., $\lceil \log_2 \mu(p_i) \rceil$), the *Pseudo-Huffman* tree is then generated by repeatedly combining the two nodes a and b with the smallest attached weights $w(a)$ and $w(b)$ to a new node c with weight $w(c) = \max(w(a), w(b)) + 1$ until a single node results. The weight of such a node $w(c)$ then gives the number of outputs required to represent all combined output patterns uniquely, i.e., one additional variable is required (aside from $\max(w(a), w(b))$) to distinguish between a and b.

Example 7. *Consider the distribution of the output patterns as shown in Table 1b. Determining the Pseudo-Huffman code starts with the nodes v_1, v_2, v_3, and v_4—one for each output pattern p_i with $\mu(p_i) > 0$. These nodes are shown at the bottom of Fig. 3. The weights are drawn inside the respective nodes. The weight of node v_1 is $w_1 = k_1 = 2$, because output pattern $p_1 = 010$ requires two garbage outputs. The weights of the nodes representing p_2, p_3, and p_4 are 1, 0, and 0, respectively. In a first step, the nodes v_3 and v_4 (both have weight 0) are combined. The resulting node v_5 has a weight of $w_5 = \max(0, 0) + 1 = 1$. Next, the two nodes with weight 1 (i.e., v_2 and v_5) are combined. The resulting node*

[5] The garbage outputs are represented by a dash, since they represent *don't care* values (as long as it is ensured that the resulting function is reversible).

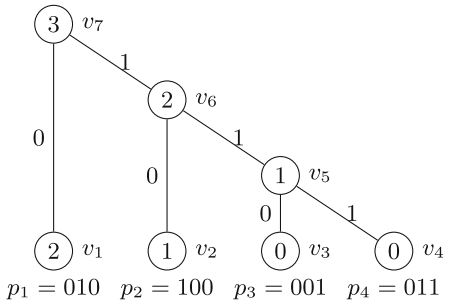


Fig. 3. Huffman tree for the function from Table 1a.

v_6 has a weight of $w_6 = \max(1, 1) + 1 = 2$. Finally, the two remaining nodes are combined to a new node v_7 with weight $w_7 = \max(2, 2) + 1 = 3$—eventually resulting in the tree shown in Fig. 3.

After generating the Pseudo-Huffman tree, the overall number of variables that are required to realise the encoded function is given by the weight of the root node of the tree. The resulting code is inherently given by the structure of the Pseudo-Huffman tree. In fact, each path from the root node to a leaf node represents a code word, where taking the left (right) edge implies a 0 (1).

Example 7 (continued)**.** *Since the root node has a weight of 3, three variables are required to realise the encoded function (without encoding,* $\max(3, 3 + 2) = 5$ *variables would be required). The path from the root node to the leaf node v_2 (which represents output pattern p_2) traverses the right edge of the root node v_7 as well as the left edge of v_6. Consequently, $c(p_2) = 10$ encodes $p_2 = 100$. Since v_2 has weight $w_2 = 1$, one output is used as garbage output in this case. Accordingly, code words for all other output patterns are determined—eventually resulting in the code shown in Table 1c. Dashes again represent* don't cares.

Following this idea, at most $n + 1$ qubits—instead of $\max(n, m + \lceil \log_2 \mu(p_1) \rceil)$—are required to embed any non-reversible function with n inputs. Concerning the design of Boolean components contained in quantum algorithms, the encoded outputs can be handled (1) *locally* where decoders are required for each sub-component that again increase the number of qubits to $\max(n, m + \lceil \log_2 \mu(p_1) \rceil)$, or (2) *globally* where subsequent components that are capable of handling encoded inputs allow remaining at $n + 1$ qubits.

Incorporating the idea of utilising coding techniques into the one-pass design flow introduced above unveils even more potential. In fact, it allows exploiting an even larger degree of freedom since the values of the garbage outputs are basically *don't care* (except the restriction that a reversible function has to be realised)—while still guaranteeing to synthesise a circuit that uses the minimum number of qubits (or even below that minimum if no decoding is required afterwards). This degree of freedom allows for synthesising circuits with significantly smaller T-count [38].

5 Mapping Quantum Circuits to NISQ Devices

In order to use currently developed *Noisy Intermediate-Scale Quantum* (NISQ) devices, the quantum algorithm to be executed has to be properly mapped to these devices such that their underlying physical constraints are satisfied (this is one part of the overall compilation task). To this end, it is assumed that the considered quantum algorithm has already been translated into a quantum circuit composed of multiple-controlled one-qubit gates. For the "quantum part" of the algorithm, this is often inherently given (e.g., by using components for which such translations are known) or done by hand. For the "Boolean part" of the algorithm, a gate-level description is often gained by *reversible circuit synthesis*, as discussed in the previous section.

Then, mapping quantum circuits to NISQ devices requires the consideration of two aspects. First, the occurring gates have to be decomposed into elementary operations provided by the target device—usually a single two-qubit gate as well as a broader variety of one-qubit gates to gain a universal gate set. Second, the *logical* qubits of the quantum circuit have to be mapped to the *physical* qubits of the target device while satisfying the so-called *coupling-constraints* given by the respective device. Since not all physical qubits are coupled directly with each other (due to missing physical connections), two-qubit gates can only be applied to selected pairs of physical qubits. Since it is usually not possible to determine a mapping such that all coupling-constraints are satisfied throughout the whole circuit, the mapping has to change dynamically. This is achieved by inserting additional gates, e.g., realising SWAP operations, in order to "move" the logical qubits to other physical ones.

While there exist several methods to address the first issue, i.e., how to efficiently decompose multiple-controlled one-qubit gates into elementary operations (see [40,41]), there is only few work on how to efficiently satisfy the coupling-constraints of real devices. Although there are similarities with recent work on nearest-neighbour optimisation of quantum circuits as proposed in [42–45], they are not applicable since simplistic architectures with 1-dimensional or 2-dimensional layouts are assumed which have a fixed coupling (all adjacent qubits are coupled) that does not allow modelling all current NISQ devices.

This section covers the mapping of the logical qubit of a quantum circuit to the physical ones of a NISQ device from a design automation perspective. Thereby, *IBM Q devices* are considered as representatives for NISQ devices to discuss the occurring challenges in detail, as well as to describe the proposed solutions. IBM's approach has been chosen, since it provides the first publicly available quantum devices (available since 2017) that can be accessed by everyone (not only academics) through cloud access. Moreover, their coupling-constraints are described more flexibly than those of other companies—allowing to map their coupling-constraints to IBM's model as well.

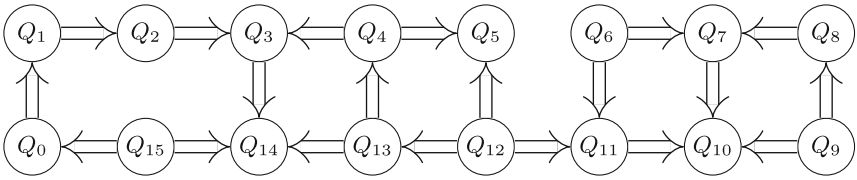


Fig. 4. IBM Q 16 Rueschlikon V1.0.0 (IBM QX3) [46].

5.1 Considered Problem

While one-qubit gates can be applied without limitations in IBM's devices, the physical architecture of the respectively developed quantum computers—usually a linear or rectangular arrays of qubits—limits two-qubit gates to neighbouring qubits that are connected by a superconducting bus resonator. In IBM's devices that use cross-resonance interaction as the basis for CNOT gates, the frequencies of the qubits also determine the direction of the gate (i.e., determining which qubit is the control and which is the target). The possible CNOT gates are captured by so-called coupling maps [46], giving a very flexible description means to specify the *coupling-constraints* of a certain quantum device. Figure 4 shows the coupling map of the IBM QX3 device. Physical qubits are visualised with nodes and a directed edge from physical qubit Q_i to physical qubit Q_j indicates that a CNOT with control qubit Q_i and target qubit Q_j can be applied.

To satisfy the coupling-constraints, one has to map the n logical qubits $q_0, q_1, \ldots, q_{n-1}$ of the decomposed circuit to the $m \geq n$ physical qubits $Q_0, Q_1, \ldots, Q_{m-1}$ of the considered quantum device such that all coupling-constraints given by the corresponding coupling map are satisfied. Unfortunately, it is usually not possible to find a mapping such that the coupling-constraints are satisfied throughout the whole circuit (this is already impossible if the number of other qubits, a logical qubit interacts with, is larger than the maximal degree of the coupling map). More precisely, the following problems—using $CNOT(q_c, q_t)$ to describe a CNOT gate with control qubit q_c and target qubit q_t, and CM to describe the edges of the device's coupling map—may occur:

- A CNOT gate $CNOT(q_c, q_t)$ shall be applied while q_c and q_t are mapped to physical qubits Q_i and Q_j, respectively, and $(Q_i, Q_j) \notin CM$ as well as $(Q_j, Q_i) \notin CM$.
- A CNOT gate $CNOT(q_c, q_t)$ shall be applied while q_c and q_t are mapped to physical qubits Q_i and Q_j, respectively, and $(Q_i, Q_j) \notin CM$ while $(Q_j, Q_i) \in CM$.

To overcome these problems, one strategy is to insert additional gates into the circuit to be mapped. More precisely, to overcome the first issue, one can insert so-called SWAP operations into the circuit that exchange of the states of two physical qubits and, by this, "move" around the logical ones—changing the mapping dynamically.

Example 8. *Figure 5 shows the effect of a SWAP gate as well as its decomposition into elementary gates supported by the IBM Q devices. Assume that the logical qubits q_0 and q_1 are initially mapped to the physical ones Q_0 and Q_1, respectively (indicated by $\rightarrow$). Then, by applying a SWAP gate, the states of Q_0 and Q_1 are exchanged—eventually yielding a mapping where q_0 and q_1 are mapped to Q_1 and Q_0, respectively.*

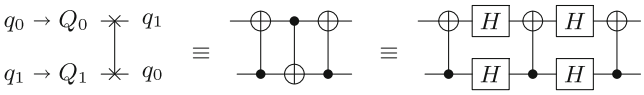
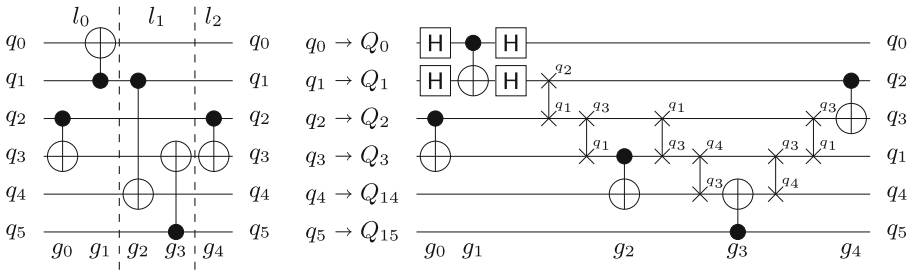


Fig. 5. Decomposition of a SWAP operation.

The second issue may also be solved by inserting SWAP operations. However, it is cheaper (fewer overhead is generated) to insert four Hadamard operations (labelled by H) as they switch the direction of the CNOT gate (i.e., they change the target and the control qubit). This can also be observed in Fig. 5, where H gates switch the direction of the middle CNOT in order to satisfy all coupling-constraints given by the coupling map (assuming that only CNOTs with control qubit Q_1 and target qubit Q_0 are possible).

However, inserting additional gates in order to satisfy the coupling-constraints drastically increases the number of operations—a significant drawback, which affects the fidelity of the quantum circuit since each gate has a certain error rate. Since each SWAP operation is composed of 7 elementary gates (cf. Fig. 5), particularly their number shall be kept as small as possible. Accordingly, this raises the question of how to derive a proper mapping of logical qubits to physical qubits while, at the same time, minimising the number of added SWAP and H operations—an $\mathcal{NP}$-complete problem as recently proven in [47,48].

Example 9. *Consider the quantum circuit composed of 5 CNOT gates shown in Fig. 6a and assume that the logical qubits q_0, q_1, q_2, q_3, q_4, and q_5 are respectively mapped to the physical qubits Q_0, Q_1, Q_2, Q_3, Q_{14}, and Q_{15} of* IBM QX3 *shown in Fig. 4 on Page 16. The first gate can be directly applied, because the coupling-constraints are satisfied. For the second gate, the direction has to be changed because a CNOT with control qubit Q_0 and target Q_1 is valid, but not vice versa. This can be accomplished by inserting Hadamard gates as shown in Fig. 6b. For the third gate, the mapping has to change. To this end, SWAP operations $SWAP(Q_1, Q_2)$ and $SWAP(Q_2, Q_3)$ are inserted to move logical qubit q_1 to become a neighbour of logical qubit q_4 (see Fig. 6b). Afterwards, q_1 and q_4 are mapped to the physical qubits Q_3 and Q_{14}, respectively, which allows applying the desired CNOT gate. Following this procedure for the remaining qubits eventually results in the circuit shown in Fig. 6b. The mapped circuit is composed of 51 elementary operations and has a depth of 36 when using a naive algorithm—a significant overhead that motivates research on improved approaches.*



(a) Original circuit. (b) Naive strategy.

Fig. 6. Mapping of a quantum circuit to IBM QX3.

5.2 Existing Approaches and Results

There exist only very few algorithms that explicitly tackle the mapping problem for IBM Q devices, and, thus, serve as alternative to IBM's own solution provided within its SDK Qiskit [49].[6] To encourage further development in this area, IBM even launched the *IBM Qiskit Developer Challenge* seeking for the best possible solution [50]. This led to the development of several approaches that explicitly consider design automation techniques to tackle the mapping problem.

The work [51] provides—for the first time—an exact approach (using a formal description of the mapping problem that is passed to a powerful reasoning engine) to solve the mapping problem by inserting the minimum number of additional H and SWAP operations. By this, a lower bound on the overhead is provided (when neglecting pre- and post-mapping optimisations), which is required to satisfy the coupling-constraints given by the quantum hardware—allowing to show that IBM's own solution often exceeds the minimal overhead by more than 100 % (even for small instances). However, the exponential nature of the mapping problem (it has been proven to be $\mathcal{NP}$-complete [47]) makes the exact approach applicable for small instances only.

This limitation—together with the fact that IBM's approach generates mapping that are far above the minimum—motivates the development of heuristic approaches. The heuristic methods presented in [52] are heuristic solution that utilises the A* search method to determine proper mappings. This allows reducing the overhead compared to Qiskit by approximately one fourth on average.[7] This difference in quality is mainly because IBM's solution randomly searches for a mapping that satisfies the coupling-constraints—leading to a rather small exploration of the search space so that only rather poor solutions are usually found. In contrast, the proposed approach aims for an optimised solution by exploring more suitable parts of the search space and additionally exploiting

[6] Note that IBM's solution randomly searches (guided by heuristics) for mappings of the qubits at a certain point of time.

[7] Note that the proposed approach has additionally been integrated into Qiskit to allow a fair comparison by utilising the same post-mapping optimisations.

information of the circuit. More precisely, a look-ahead scheme is employed that considers gates that are applied in the near future and, thus, allows determining mappings which aim for a global optimum (instead of local optima) with respect to the number of SWAP operations.

Even though this heuristic approach allows outperforming Qiskit's mapping algorithm, it has some scalability issues when used for mapping certain random circuits for validating quantum computers [19], which also served as benchmarks in the *IBM Qiskit Developer Challenge* (a challenge for writing the best quantum-circuit compiler to encourage development). These circuits provide a worst-case scenario that heavily affects the efficiency of the proposed heuristic approach. Therefore, a dedicated approach is proposed in [53], which explicitly considers their structure by using dedicated pre- and post-mapping optimisations. The resulting methodology has been declared as winner of the IBM Qiskit Developer Challenge, since it generated mapped/compiled circuits with at least 10 % lower costs than the other submissions while generating them at least 6 times faster, and is currently being integrated into Qiskit by researchers from IBM. Besides that, all mapping approaches developed in context of this thesis are publicly available at http://iic.jku.at/eda/research/ibm_qx_mapping.

6 Conclusion

This chapter has shown the great potential of bringing knowledge gained from the design automation of conventional circuits and systems into the quantum realm. More precisely, quantum-circuit simulation, the design of Boolean components for quantum algorithms, as well as technology mapping have been considered from a design automation perspective—leading to improvements of several orders of magnitude (with respect to runtime or other design objectives) in many cases. For further information on the developed algorithms we refer to the cited papers. In the future, this development shall continue on a larger scale—eventually providing the foundation for design automation methods that accomplish for quantum computing what the design automation community realised for conventional (electronic) circuits.

Acknowledgments. This work has partially been supported by the European Union through the COST Action IC1405 and the LIT Secure and Correct System Lab funded by the State of Upper Austria.

References

1. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information. AAPT **70**, 558 (2002)
2. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Symposium on the Theory of Computing, pp. 212–219 (1996)
3. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)

4. Montanaro, A.: Quantum algorithms: an overview. npj Quantum Inf. **2**, 15023 (2016)
5. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
6. Coles, P.J., et al.: Quantum algorithm implementations for beginners. arXiv preprint arXiv:1804.03719 (2018)
7. Gambetta, J.M., Chow, J.M., Steffen, M.: Building logical qubits in a superconducting quantum computing system. npj Quantum Inf. **3**(1), 2 (2017)
8. Kelly, J.: A preview of Bristlecone, Google's new quantum processor (2018). https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html
9. Hsu, J.: CES 2018: Intel's 49-qubit chip shoots for quantum supremacy. IEEE Spectrum Tech Talk (2018). https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy
10. Sete, E.A., Zeng, W.J., Rigetti, C.T.: A functional architecture for scalable quantum computing. In: International Conference on Rebooting Computing (ICRC), pp. 1–6 (2016)
11. IonQ: IonQ: trapped ion quantum computing. https://ionq.co. Accessed 15 June 2019
12. Nay, C.: IBM unveils world's first integrated quantum computing system for commercial use. https://newsroom.ibm.com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercial-Use. Accessed 15 June 2019
13. Horsman, C., Fowler, A.G., Devitt, S., Van Meter, R.: Surface code quantum computing by lattice surgery. New J. Phys. **14**(12), 123011 (2012)
14. Gottesman, D.: An introduction to quantum error correction and fault-tolerant quantum computation. In: Quantum Information Science and Its Contributions to Mathematics, Proceedings of Symposia in Applied Mathematics, vol. 68, pp. 13–58 (2010)
15. Yamashita, S., Markov, I.L.: Fast equivalence-checking for quantum circuits. In: International Symposium on Nanoscale Architectures. pp. 23–28. IEEE Press (2010)
16. Niemann, P., Wille, R., Drechsler, R.: Equivalence checking in multi-level quantum systems. In: International Conference of Reversible Computation, pp. 201–215 (2014)
17. Burgholzer, L., Wille, R.: Improved DD-based equivalence checking of quantum circuits. In: Asia and South Pacific Design Automation Conference (ASP-DAC) (2020)
18. Boixo, S., et al.: Characterizing quantum supremacy in near-term devices. Nat. Phys. **14**(6), 595 (2018)
19. Cross, A.W., Bishop, L.S., Sheldon, S., Nation, P.D., Gambetta, J.M.: Validating quantum computers using randomized model circuits. arXiv preprint arXiv:1811.12926 (2018)
20. Smelyanskiy, M., Sawaya, N.P.D., Aspuru-Guzik, A.: qHiPSTER: the quantum high performance software testing environment. arXiv preprint arXiv:1601.07195 (2016)
21. Zulehner, A., Wille, R.: Advanced simulation of quantum computations. IEEE Trans. CAD Integr. Circuits Syst. **38**, 848–859 (2019)
22. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Quantum Circuit Simulation. Springer, Dordrecht (2009). https://doi.org/10.1007/978-90-481-3065-8

23. Niemann, P., Zulehner, A., Wille, R., Drechsler, R.: Efficient construction of QMDDs for irreversible, reversible, and quantum functions. In: Phillips, I., Rahaman, H. (eds.) RC 2017. LNCS, vol. 10301, pp. 214–231. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59936-6_17
24. Zulehner, A., Wille, R.: Matrix-vector vs. matrix-matrix multiplication: potential in DD-based simulation of quantum computations. In: Design, Automation and Test in Europe, European Design and Automation Association (2019)
25. Zulehner, A., Hillmich, S., Wille, R.: How to efficiently handle complex values? Implementing decision diagrams for quantum computation. In: International Conference on CAD (2019)
26. Niemann, P., Datta, R., Wille, R.: Logic synthesis for quantum state generation. In: International Symposium on Multi-Valued Logic, pp. 247–252. IEEE (2016)
27. Niemann, P., Wille, R., Drechsler, R.: Improved synthesis of Clifford+T quantum functionality. In: Design, Automation and Test in Europe, pp. 597–600 (2018)
28. Zulehner, A., Niemann, P., Drechsler, R., Wille, R.: Accuracy and compactness in decision diagrams for quantum computation. In: Design, Automation and Test in Europe (2019)
29. Hillmich, S., Zulehner, A., Wille, R.: Concurrency in DD-based quantum circuit simulation. In: Asia and South Pacific Design Automation Conference (ASP-DAC) (2020)
30. Zulehner, A., Hillmich, S., Markov, I., Wille, R.: Approximation of Quantum States Using Decision Diagrams. Asia and South Pacific Design Automation Conference (ASP-DAC) (2020)
31. Ekert, A., Jozsa, R.: Quantum computation and Shor's factoring algorithm. Rev. Mod. Phys. **68**(3), 733 (1996)
32. Soeken, M., Roetteler, M., Wiebe, N., De Micheli, G.: LUT-based hierarchical reversible logic synthesis. IEEE Trans. CAD Integr. Circuits Syst. **38**, 848–859 (2018)
33. Zulehner, A., Wille, R.: Make it reversible: efficient embedding of non-reversible functions. In: Design, Automation and Test in Europe, European Design and Automation Association, pp. 458–463 (2017)
34. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of reversible circuits with minimal lines for large functions. In: Asia and South Pacific Design Automation Conference, pp. 85–92 (2012)
35. Zulehner, A., Wille, R.: Improving synthesis of reversible circuits: exploiting redundancies in paths and nodes of QMDDs. In: Phillips, I., Rahaman, H. (eds.) RC 2017. LNCS, vol. 10301, pp. 232–247. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59936-6_18
36. Zulehner, A., Wille, R.: One-pass design of reversible circuits: combining embedding and synthesis for reversible logic. IEEE Trans. CAD Integr. Circuits Syst. **37**(5), 996–1008 (2018)
37. Zulehner, A., Wille, R.: Skipping embedding in the design of reversible circuits. In: International Symposium on Multi-Valued Logic, pp. 173–178. IEEE (2017)
38. Zulehner, A., Wille, R.: Exploiting coding techniques for logic synthesis of reversible circuits. In: Asia and South Pacific Design Automation Conference, pp. 670–675. IEEE Press (2018)
39. Zulehner, A., Niemann, P., Drechsler, R., Wille, R.: One additional qubit is enough: encoded embeddings for Boolean components in quantum circuits. In: International Symposium on Multi-Valued Logic (2019)

40. Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(6), 818–830 (2013)
41. Miller, D.M., Wille, R., Sasanian, Z.: Elementary quantum gate realizations for multiple-control Toffoli gates. In: International Symposium on Multi-Valued Logic, pp. 288–293. IEEE (2011)
42. Wille, R., Keszocze, O., Walter, M., Rohrs, P., Chattopadhyay, A., Drechsler, R.: Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In: Asia and South Pacific Design Automation Conference, pp. 292–297 (2016)
43. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In: Design Automation Conference, pp. 41–46 (2013)
44. Wille, R., Quetschlich, N., Inoue, Y., Yasuda, N., Minato, S.I.: Using πDDs for nearest neighbor optimization of quantum circuits. In: International Conference of Reversible Computation, pp. 181–196 (2016)
45. Zulehner, A., Gasser, S., Wille, R.: Exact global reordering for nearest neighbor quantum circuits using A*. In: Phillips, I., Rahaman, H. (eds.) RC 2017. LNCS, vol. 10301, pp. 185–201. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59936-6_15
46. IBM Q team: IBM Q 16 Rueschlikon backend specification v1.0.0. https://ibm.biz/qiskit-rueschlikon. Accessed 15 June 2019
47. Botea, A., Kishimoto, A., Marinescu, R.: On the complexity of quantum circuit compilation. In: Symposium on Combinatorial Search (2018)
48. Siraichi, M., Dos Santos, V.F., Collange, S., Pereira, F.M.Q.: Qubit allocation. In: International Symposium on Code Generation and Optimization (CGO), pp. 1–12 (2018)
49. Cross, A.: The IBM Q experience and QISKit open-source quantum computing software. Bull. Am. Phys. Soc. **63**(1) (2018)
50. IBM Q team: QISKit Developer Challenge. https://qx-awards.mybluemix.net/#qiskitDeveloperChallengeAward. Accessed 15 June 2019
51. Wille, R., Burgholzer, L., Zulehner, A.: Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In: Design Automation Conference (2019)
52. Zulehner, A., Paler, A., Wille, R.: An efficient methodology for mapping quantum circuits to the IBM QX architectures. IEEE Trans. CAD Integr. Circuits Syst. **38**, 1226–1236 (2018)
53. Zulehner, A., Wille, R.: Compiling SU(4) quantum circuits to IBM QX architectures. In: Asia and South Pacific Design Automation Conference, pp. 185–190. ACM (2019)